6. PC-Based Applications

This chapter describes the DII COE features that are available for Personal Computer (PC) platforms. The present DII COE supports PC Windows NT[1] only. The COE concept is not specific to Unix, or NT, or any other operating system or windowing environment. However, certain adjustments to COE implementation details are required to support differences between the PC-based NT environment (use of '\' versus '/' in naming directories, byte swapping, etc.) and Unix, as well as to take advantage of features offered in the NT environment (e.g., registry).

The extensions described in this chapter to accommodate NT are not platform-dependent (e.g., limited to 80x86 PCs). Commercial industry has implemented the Microsoft NT operating system on selected other platforms (e.g., DEC), but such platforms are not presently in wide use in the DII community. COE support for NT on platforms other than PCs will be considered when they are in widespread use in the DII community. Throughout this version of the *I&RTS*, NT and PC may be used interchangeably with the understanding that NT is not limited to PC platforms.

This chapter begins by discussing approaches to providing a PC-based COE. The situation is complicated by the need to address the concept of a "universal desktop." That is, the ability to access Unix applications from a PC platform or vice versa. System designers may find the techniques described useful for Windows 3.1 and Windows 95 platforms, but they are not included in the list of DII-supported platforms for reasons already noted.

[1] Windows 3.1 and Windows for Workgroups 3.11 are not supported. Windows 95 is not presently a supported platform because of known security problems within the operating system. When the security problems are resolved, Windows 95 may be added to the list of supported platforms.

6.1 Approaches for a PC-Based COE

Many operational sites have a large investment in PCs and PC software, or are planning significant PC acquisitions. Introduction of a Unix-based COE system poses challenges for such sites in how to best reuse existing computing assets to minimize capital, training, maintenance, administration, and other related costs. The most difficult technical challenge is that some functions are effectively operating system specific because they may only exist for one operating system (e.g., access to the JOPES database), because of strong operator preference for tools available for one operating system environment over those available in another environment, or because of the availability of alternatives from the commercial market (e.g., spreadsheets, briefing support).

One approach is to implement a homogeneous network, or to divide work between operators in such a way that any individual operator deals with only those functions supported by a single operating system. This approach is not practical because the "optimum" platform to use is frequently application-specific. Imposing an organization or dictating work responsibilities to accommodate system limitations is likewise undesirable and impractical.

A second approach is to provide operators with both a Unix platform and a PC platform. This approach is often neither viable nor desirable due to cost considerations, or the lack of available desk space for two monitors and keyboards. Moreover, data sharing between the two systems is difficult or impossible for even simple operations such as cut and paste.

Neither approach is satisfactory, but this problem is not unique to DISA programs or the DII COE concept. It has been recognized in commercial industry as well where operators and administrators must manage and use a heterogeneous network. The solution proposed by commercial industry and adopted by the DII COE is the concept of a *Universal Desktop*. A Universal Desktop is capable of utilizing resources of the entire network, regardless of vendor platform, while displaying and accessing the information from a single computer monitor, keyboard, and mouse. Advances in distributed client/server technology and advances in windowing software provide the foundation necessary to realize the Universal Desktop concept. The Universal Desktop concept is maturing, but there are not yet any commercially available products that have widespread acceptance. There are several distinct approaches to the Universal Desktop, each with its own set of advantages, limitations, and advocates.

In the COE context, an operator on a Unix platform needs to have access to native Unix applications as well as selected PC-based office automation applications. An operator on a PC platform needs to have access to native Windows applications, and a selected number of Unix-based applications. This subsection briefly describes four Universal Desktop approaches applicable to the COE:

1. Provide remote execution of client applications on a server, but local display of results;

2. Perform software emulation to run applications locally;

3. Create a Web interface to applications running on a server, but with local display of results in a Web browser; and

4. Perform native execution of client applications through porting as necessary for selected applications.

These approaches are described from least recommended to most recommended. There are several commercially available solutions for the first two approaches that are described below. The DII COE does not encourage either approach for reasons that will be described below. The COE provides a Web-based solution for the third approach that is described briefly here, but more fully in Chapter 7. The third and fourth approaches are the preferred COE methods. Chapter 8 describes DCE capabilities provided within the COE, which is one way to achieve the objectives of both the first and fourth approach. A non-DCE alternative is discussed below in subsection 6.1.4 that may be suitable for legacy applications that are not presently designed to use DCE.

It is difficult to discuss approaches for a PC-based COE without mentioning particular COTS products. Discussion of a COTS product should not be taken as DII endorsement of the product, nor should failure to mention a particular product be taken as a negative endorsement for the product. The products mentioned are described only to illustrate commercially available approaches or technologies involved in a PC-based COE. COTS software and licenses are the responsibility of the site or the cognizant program manager for any required software, regardless of the approach selected. Sites or program managers who elect to use commercial products may require a combination of vendor products if access is required from both PC and Unix platforms.

6.1.1 Remote Execution

Remote execution is the ability to run an application on a platform different from the operator workstation. If the application creates any displayable results, the results are displayed locally on the operator's workstation. If the remote platform uses the same operating system and windowing software as the local platform, the issues are relatively straightforward and are not discussed in detail in this chapter. When the remote platform and the local platform do *not* use the same operating system and windowing software, the technical challenges are more difficult.

Ideally, the results should be displayed using the same "look and feel" as the native windowing software. That is, if an NT workstation remotely executes a Unix client application, the results should be displayed in the Microsoft Windows NT "look and feel." Unfortunately, this is usually not possible because the client application is written for a specific windowing environment, while the APIs and communications protocols are significantly different between X Windows and Microsoft Windows.

There are two basic approaches to providing remote execution: providing a ported "window server" on the local machine while executing the application remotely, and distributed computing techniques.

6.1.1.1 Window Server Approach

Both Microsoft Windows and X Windows are designed around a client/server paradigm. An application program, the client, sends request to the windowing software, the server, to display text or graphics. There is no fundamental theoretical requirement that the client and server reside on the same physical platform, although that is the most common configuration. If the operator is using a machine that hosts the windowing server, then the fact that applications are being executed remotely or locally is usually transparent.

Figure 6-1 shows how it is possible to exploit this feature of windowing servers to achieve remote execution of an application with local display of the results. In this scheme, a workstation has native applications that are accessed through the native desktop and displayed through the native windowing software. Remote applications are initiated by the native desktop, but are executed remotely on an Application Server. Display commands generated by the remote application are routed back across the LAN to a window server (labeled as Foreign Window Server in Figure 6-1) running on the local machine that requested the application. The Foreign Window Server then interfaces with the Native Window Server to achieve the resulting display.

For example, assume that the operator workstations in Figure 6-1 are PCs and that it is desired to access Unix applications running on the Application Servers. The Unix application will create display commands using the X protocol that are then routed back to the PC. The display commands are received by an X server running on the PC and translated to native Microsoft Windows commands that are then displayed. An operator at the PC can still access applications local to the PC. This provides access to both PC and Unix applications from the same desktop in a way that is generally transparent to the user.

A variety of commercial products are available that implement variations on this basic theme. Product differentiation falls into essentially the following categories:

- whether the applications server is a Unix or NT platform (i.e., whether the goal is to access Unix applications from a PC or to access PC applications from a Unix workstation);

- whether or not remote applications are integrated with the native desktop or require a separate desktop for remote applications; and

- implementation efficiencies in sending display commands from the application to the Foreign Window Server.

The DII COE provides support for the Window Server approach, but does not recommend it because each commercial product[2] suffers from one or more of the following drawbacks:

[2] *Every* commercial product in the list presented has some drawbacks, but a particular product does not necessarily suffer *all* of the drawbacks presented. Each product is designed for a particular environment and operational metaphor and thus may work better when used in one way but not in another. With the

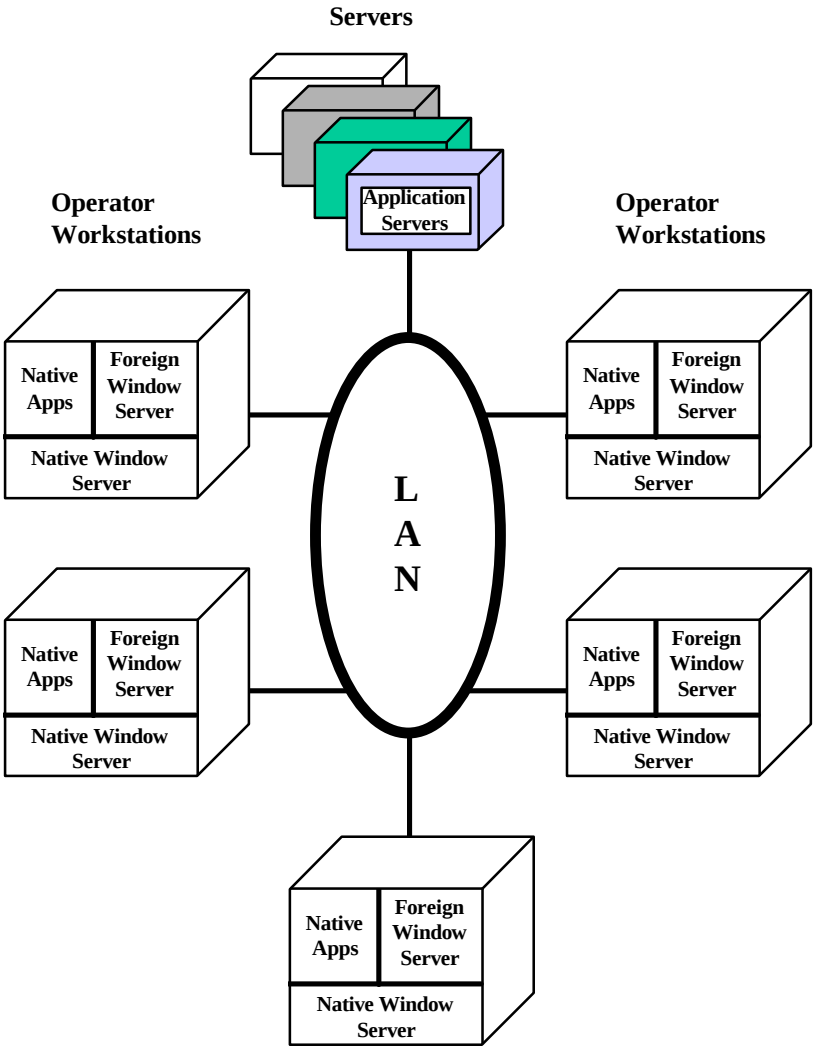

Figure 6-1: Remote Execution with Window Server Port

- increased LAN loading;
- decreased responsiveness, especially when applications make heavy use of the mouse or graphics (screen savers should usually be disabled);
- increased load on the application servers;

lack of a true industry standard approach, it is not possible to present a single COTS product that satisfies a large majority of the DII COE community.

- limited number of simultaneous users supported by an individual application server (note that the limit may actually be determined by the number of *applications* being run at one time, not the actual number of *users*);

- increased difficulty in restricting or auditing access to applications, or creating user profiles;

- the requirement for two GUI interfaces (X Windows and Microsoft Windows) on the same platform;

- the requirement for a modified version of NT (this means that NT upgrades from Microsoft may not work on the modified platform);

- potential cost of additional hardware requirements, especially PC systems;

- potential compatibility problems between applications available under Windows 3.1, Windows 95, and Windows NT;

- no access to both native and foreign applications from the same platform within the same desktop; and

- limited data sharing at best between PC Windows and X Windows (e.g., cut and paste).

6.1.1.1.1 Accessing Unix Applications from a PC

Products such has Hummingbird eXceed® and AGE Xoftware32® provide X servers that run on a PC in several Microsoft Windows environments. These products allow PC workstations to remotely access Unix applications. Modifications to client applications are not normally required for this approach. However, this approach alone does not provide access to PC applications from a Unix workstation.

The COE provides an account group, `RemoteX`, which contains the necessary files to establish an appropriate runtime environment on the Unix platform. The `DISPLAY` environment variable is automatically set and software is in place to limit the number of users who may access the Unix applications server at one time. User profiles and login accounts are administered from the Unix platform. The Unix desktop software presents the same interface to the PC-based user as it does to the Unix-based user and provides the same level of auditing and security as for Unix-based users.

Hardware requirements for this approach are dependent upon which PC X server software is selected. Refer to specifications provided by the vendor. Procurement and installation of the appropriate PC X server and associated network software are the responsibility of the site or cognizant program manager.

6.1.1.1.2 Accessing PC Applications from Unix

WinDD® is based on Windows NT and was developed by a commercial vendor under a licensing arrangement with Microsoft. WinDD is completely compatible with NT, but extends NT to support multiple users and display of a complete Windows NT desktop within an X window. Figure 6-2 illustrates the approach.

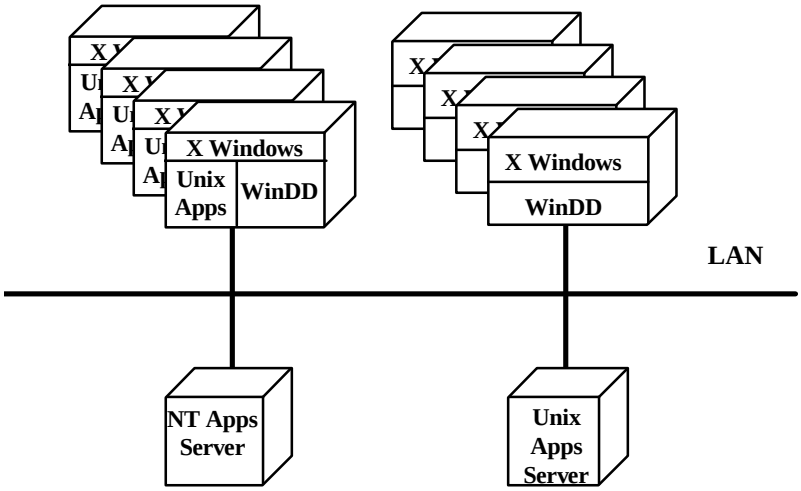

Figure 6-2: WinDD Approach

With WinDD, PC applications are loaded on one or more NT-based application servers, while Unix applications are loaded on one or more Unix-based application servers. Some of the Unix applications may also be distributed or duplicated on sufficiently powerful Unix workstations. Alternatively, the workstations may be simply X servers. Unix workstations access Unix applications via X Windows protocols across the LAN, or through native execution, while PC applications are accessed on the Unix workstation through WinDD software. WinDD is loaded on both the NT applications server and individual workstations. On the workstations, NT applications appear within an X window controlled by WinDD software. PC workstations may access PC applications either locally or from the server, but no access is provided to Unix applications from the PC workstations.

Sites may purchase WinDD from DISA for Unix platforms. DISA provides WinDD packaged as a COTS segment for simplified installation and it is already configured to properly run on Unix workstations within the COE environment. The site must purchase and install WinDD for the NT server from the commercial vendor and is responsible for WinDD software licenses. Application software running on the NT server is also the responsibility of the site.

Refer to vendor-supplied specifications for a complete set of hardware requirements to host WinDD software.

6.1.1.1.3 Accessing Applications from a Single Desktop

NTED® and TED for Windows® are two commercial products that are based on Common Desktop Environment (CDE) technology. NTED software runs on a Unix platform and provides access to NT applications from within the CDE desktop. The appearance to the operator is a seamless environment incorporating both NT and X Windows applications from the same Unix workstation. NTED requires that a modified version of NT be installed on an NT applications server. The modified version of NT was developed under a marketing agreement with Microsoft.

TED for Windows runs on PC workstations and provides seamless access to PC and X Windows applications. In essence, this product puts a CDE desktop on the PC with the associated advantages (uniform desktop across PCs and Unix, virtual workspace management, etc.) and disadvantages (different environment than native PC).

Refer to vendor-supplied specifications for a complete set of hardware requirements to host TED for Windows and NTED software.

6.1.1.2 Distributed Computing Approach

Distributed computing technologies such as sockets, DCE, CORBA, and DCOM/OLE provide another approach to remote execution. Chapter 8 describes the COE extensions to support DCE while support for distributed objects is scheduled for a later *I&RTS* release.

The DII COE provides support for DCE Remote Procedure Calls and sockets to allow applications running on one platform to access and execute functions on another platform. This approach avoids the shortfalls from the previous subsection, and has the advantage that load balancing is more easily achieved. However, this approach requires that both the client application and the remote function be designed and implemented with such a capability in mind. Since such modifications are not generally practical for existing applications without significant modifications, this approach is mostly useful for new development efforts, or for legacy applications that are being migrated/ported to the DII COE.

6.1.2 Software Emulation

Another approach to achieving a Universal Desktop is to provide a software emulator that executes “foreign” code on the local machine. JAVA is a recent technological advance that appears promising, but the technology is not sufficiently mature yet to ensure that security problems, including viruses, will not be a problem. Moreover, JAVA will require applications to be rewritten to take advantage of any JAVA capabilities. Thus, it is only suitable for consideration for new applications.

Commercially available products that perform emulation are typically designed to emulate the Intel 80x86 instruction set on a Unix platform. One such product, SunSoft Wabi®, allows Windows applications to be loaded and executed directly on Unix platforms. The product translates, at runtime, calls to Microsoft Windows APIs to the equivalent

X Windows APIs and emulates 80x86 instructions. This approach provides an operator access to Unix and PC applications, but without the need to have a PC platform. However, emulation on a Unix platform is slower than native execution on the PC, and the approach does not provide access to Unix applications from a PC.

Sites and program managers who elect to take this approach are cautioned that not all Windows products are supported under Wabi. Products that require specialized device drivers (sound cards, video drivers, etc.) may not operate correctly. Products that require MS-DOS® either for installation or operation require purchase of an MS-DOS emulator that is not provided with Wabi.

Wabi requires the installation of Windows onto a Unix machine. Sites may purchase Wabi and Windows directly from DISA. DISA provides Wabi and Windows pre-packaged as COTS segments for simplified installation and they are already configured to properly run on Unix workstations within the COE environment. The site is responsible for software licenses and any additional Windows application software.

Refer to vendor product specifications for the most current list of Wabi-compliant products. Also, vendor specifications should be consulted for Wabi hardware requirements.

DISA provides support for Wabi, but does not recommend it due to:

- a limited number of supported Windows applications,
- the sluggish performance of emulation approaches in general,
- the requirement to load Windows as well as Wabi,
- limited support for device drivers, and
- potential security holes in mixing Windows and Unix.

6.1.3 Web Access

Web technologies make it feasible to provide a reasonably powerful form of remote execution by simply providing a Web browser on the local machine. This is an attractive approach because it is much simpler than the previous two approaches (subsections 6.1.1 and 6.1.2), and it has the significant advantage that it works acceptably when the local and target platform must communicate over relatively low bandwidth communications channels. This approach requires no code modifications on the local machine and it is a hardware independent approach. It only requires the installation of a Web browser. That is, if an applications server is created which provides Web access, a browser running on any machine can access the applications.

The DII COE supports Web-based approaches. Extensions to support Web-based applications are described further in Chapter 7. This approach to a Universal Desktop offers the following advantages:

- the approach is hardware independent for the client;
- acceptable performance is usually available even across low bandwidth channels;
- no code modifications are required on the client workstation (e.g., the applications are run remotely, or locally if the application is available locally and designed to use a Web browser interface);
- application software updates are performed on the Web server and become immediate for all clients as soon as the client reconnects; and
- the approach is very useful for documentation because documents stored in HTML format can be readily accessed anywhere by any user with a browser.

This approach suffers from the following disadvantages:

- the approach does not work for legacy applications;
- GUI navigation is more cumbersome than a menu/icon approach, and it is more difficult to find the desired application;
- update rates for applications are generally much slower than for native execution or other approaches (this may not adversely affect certain applications, such as database queries, where the display is relatively static); and
- restricting and monitoring access to specific features is more difficult.

6.1.4 Native Execution

The preferred COE approach to the Universal Desktop is to provide native execution of applications on each supported platform. Figure 6-3 illustrates the approach. Servers[3] are hosted on whichever platform is most convenient. Client applications on the Unix platform use X Windows for GUI support and access COE servers through a set of APIs. NT client applications use Microsoft Windows for GUI support and also access servers through APIs. If the requested service is on the same platform as the client, the request goes directly to the server. If the service requested is not available on the same platform as the requesting client, the request is routed across the LAN, through the requested server's APIs, and hence to the appropriate server. Whether or not the server is running on the

[3] The general approach described here is the preferred approach. However, at the time this document was written, existing servers within the COE are available only on Unix platforms. The strategy is to provide a set of APIs on the NT platform that are identical to those found on the Unix platform. Requests from NT clients will be routed to the appropriate Unix server and results passed back to the client. As required, servers will be ported to NT platforms over time so that the end result is a set of servers that are available for both Unix and NT platforms. APIs will be preserved so that applications that presently run on either a Unix or NT platform will not require porting just because the server has moved from a Unix to an NT platform or vice versa.

same type of platform as the requesting client is not an issue because the APIs handle the request without the need to know what type of platform is making the request. Network byte order (see subsection 6.6.6) is used to ensure that byte swapping is not a problem. Responses are passed back from the server, across the LAN if necessary, to the application as a response from APIs.

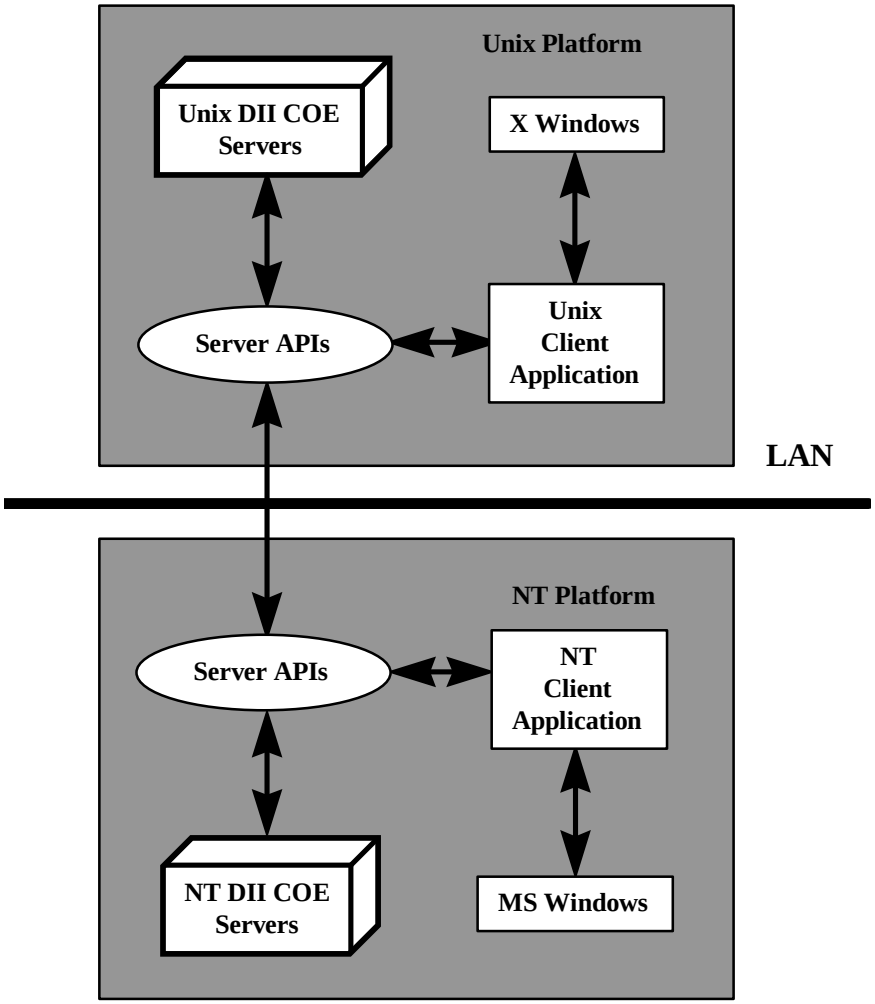

Figure 6-3: Native Execution

Note: The approach shown is not specific to PCs, but is general and will work for different operating environments. NT and Unix are illustrated here because that is the immediate concern for the DII COE. Initial implementation will be on Unix and PC NT platforms.

There are several advantages to this approach:

- Client applications use the same set of APIs whether executing on a Unix or an NT platform;

- Operators have a single desktop using the native GUI support (e.g., X Windows for Unix platforms, Windows for NT platforms);

- No special programming or site administration techniques are required;

- No additional software products must be purchased; and

- Sites may allocate workstations by balancing availability, cost, preference, and operator workload as appropriate.

The remainder of this chapter describes the PC-based COE from the perspective of native execution. No specialized hardware or programming techniques are required. Applications to be accessible from the PCs are loaded directly on PC workstations. Sites will utilize Unix platforms as appropriate for database servers and as servers for critical COE components (message handling, comms, etc.). Sites may provide operators with Unix workstations or PC workstations as appropriate and as determined by operator work requirements.

6.2 Disk Directory Structure

The NT-based COE uses the same basic directory structure shown in related figures from Chapter 5. However, Intel-based computers store data bytes in a different order than other processors. This makes data sharing via disk more difficult. This section describes the COE disk directory extensions required to support PCs.

> **Note:** The PC file system allows a disk drive to be designated (e.g., `C:`, `D:`). The COE *allows* a disk drive to be designated, but does not *require* a disk drive to be designated. In each case where a disk drive is given, it is assumed to be advisory only because segments must work in environments where the number of disk drives and their names is not known. (Although it would be unusual, it is possible to set up a PC where the first hard disk drive is not drive `C:`.) When a disk drive designation is given, the COE will attempt to satisfy whatever request is given using the drive specified. If the request cannot be satisifed, the COE will try all remaining hard disk drives until the request is satisfied or all drives have been tried unsuccessfully.

Basic Directory Structure

The logical directory structure shown in Chapter 5 is preserved for PCs. On disk drive `C`, subdirectory `\h` is created at the root level with subdirectories `COTS`, `AcctGrps`, `COE`, `data`, etc. Unless overridden by the installer, the installation software will attempt to put segments on disk drive `C` first. If it cannot do so, it will load the segment on the next available hard disk. The installation software will create the directory `\h` and other subdirectories as required on additional hard disk drives. The installer software will distribute segments across multiple hard disks, either by user selection or based on available disk space. Once a segment is loaded, a segment can determine its home and data directories through the appropriate *Preferences* APIs.

The environment variable `INSTALL_DIR` is set to point to where the segment was loaded at install time, just as for Unix platforms, and includes the disk drive designation in the pathname. At runtime, `COEFindDrive` (see Appendix C) can be used to find which disk drive contains the requested segment.

> **Note:** The installation software will *not* consider network disk drives in looking for the next available hard disk. Segments can only be installed on local hard disks, not floppies or network drives.

Segment Directory Structure

A `Scripts` subdirectory is optional for NT segments because environment extension files are not supported, nor are they needed. Account group segments that need to establish global environment settings shall do so by entering required settings in the

registry. Segments that need to establish local environment settings may do so through a `.INI` file that shall be located in the segment's `data\INI` subdirectory. All of a segment's private INI files shall be stored in the segment's `data\INI` subdirectory.

NT segments shall place all executables in the `bin` subdirectory. Segments that contain dynamic link libraries (DLL files) shall also place them in the `bin` subdirectory. Except for COTS segments, segments are not allowed to load DLL files in any other subdirectory.

USERS Directory Structure

The NT COE uses the same operator directory structure as the Unix COE, as described in Chapter 5. Local operator accounts are specific to a single NT workstation, while global operator accounts are accessible from any NT PC on the network. However, operator accounts may not be mixed between Unix and NT platforms. Thus, an operator account, whether global or local, is either an NT operator account or a Unix operator account, but never both.

Global operator account subdirectories (e.g., `\h\USERS\global`) are physically located on an NT designated as the server. This directory is made accessible to other PCs on the network through the `share` command.

Environment variables `USER_HOME`, `USER_DATA`, and `USER_PROFILE` are set by the appropriate account group and have the meaning described in Chapter 5. They are provided for backwards compatibility and should not be used in the NT-based COE. As with Unix applications, segments shall use a *Preferences* API to locate user-related data. This is because data may ultimately be moved to the registry or reside in different locations depending upon the NT configuration (e.g., workgroups versus domains). By using the *Preferences* APIs, the developer can assure future compatibility.

Data Directory Structure

Chapter 5 defines data in terms of data scope. Local data is stored underneath `\h\data\local` while global data is stored underneath `\h\data\global`. Because data stored on the PC is not directly compatible with Unix platforms, an additional data subdirectory is created for storing PC *only* global data. This is the subdirectory `\h\data\PCglobal`. Segments shall follow the same rules for this directory as for the `\h\data\global` directory, except that only PC segments are allowed to access it. This subdirectory is physically located on a PC designated as the server and made accessible to other PC workstations on the network.

Like global data, `PCglobal` data is shared between workstations. However, `PCglobal` data (and local data on PC workstations) is stored in native PC-byte order and can only be shared among PCs. PCs may also access data stored in the `\h\data\global` subdirectory. However, this directory is *always* physically located on a Unix machine designated as a server. PC segments shall read and write data in the `\h\data\global`

directory in network byte order. PC segments shall read and write data in the `\h\data\local` and `\h\data\PCglobal` directories in native PC byte order.

The installation software attempts to load data segments on the `C` disk drive. If there is insufficient room, it will load the data on the next available hard disk drive. PC segments must use the `COEFindData` tool (see Appendix C) to determine where data is actually stored.

Miscellaneous

1. Segments shall use file extensions that correspond to conventional Windows usage. That is, use `.EXE` for executables, `.DLL` for dynamic link libraries, `.TXT` for ASCII text files, etc. Note this means that NT segment descriptor files **should** use the `.TXT` extension,[4] but **shall** use the `.BAT` (for batch[5] files) or `.EXE` (for compiled programs) extension for `PostInstall`, `DEINSTALL`, `PreInstall`, and `PreMakeInst`.

2. Segments, excepting COTS segments, shall not set the Windows `path` environment variable.

3. Segments shall use the standard Windows APIs to locate a directory for temporary disk storage. This corresponds to using `/tmp` in Unix. Segments shall delete temporary files when an application terminates. Unlike the Unix-based COE, the NT-based COE does *not* automatically delete files in the Windows temporary directory when the computer is rebooted. This is in keeping with current commercial usage of the Windows temporary directory.

4. Segments shall not add a global "home" environment variable to the affected account group. Segments shall use `COEFindDrive` to determine the location of a segment after it has been loaded.

5. Environment extension files are neither supported nor required in the NT-based COE.

[4] For backwards compatibility, NT segments may omit the `.TXT` extension. However, this is strongly discouraged. The segment must be consistent in either *always* using the `.TXT` extension or *never* using it. `VerifySeg` will strictly fail a segment that does not follow this convention. Otherwise it will be confusing and unclear which descriptor takes precedence when a segment includes the same segment descriptor, once with the `.TXT` extension and once without it.

[5] Developers should avoid the use of batch files and use executables whenever possible. Batch files, in PC NT, will cause a command shell window to pop us while the batch file is running.

6. `app-defaults` subdirectories are not meaningful in the NT-based COE. Special handling of fonts (i.e., a `fonts` subdirectory) is not currently supported in the NT-based COE, but may be in the future. NT segments should not include either of these subdirectories. If they are included with a segment, the installation tools will not do any special processing for these subdirectories as they do for the Unix-based COE.

6.3 Account Groups

Account groups in the NT-based COE correspond to Windows Program Groups. The present NT COE does not include the `CharIF` or `DBAdm` account groups.

When the COE is loaded, the installation tools create program groups `SecAdm` and `SysAdm`. The program items in each program group are determined as segments are loaded. Some program items, specifically for `SecAdm` and `SysAdm`, are provided by native Windows software and therefore will also be found in other program groups provided by Windows. This is done by creating duplicate icons that point to the same executable, not by creating multiple copies of the software.

As with the Unix COE, the specific icons and program groups available to an operator depend upon the operator profile.

6.4 Registry Usage

Microsoft Windows programs have traditionally created "INI" files to store configuration information. Windows 95 and Windows NT use a *registry*[6] instead to store hardware parameters, configuration data, and Windows-maintained operator preferences. The registry is structured as a hierarchical database of keys organized into a tree structure.

NT segments should not overuse the Windows registry in place of INI files. In particular, operator preferences that are very segment specific should not be stored in the registry since this may needlessly fill up the registry, and it will be difficult manage as user accounts are created and removed. Moreover, the registry is not portable between NT and Unix. It is recommended that operator preferences be stored underneath `\h\USERS` to minimize porting problems between Unix and NT applications. (Use the appropriate COE APIs to determine the correct data directory for the current operator.) Segments may use private INI files but if they are used, they shall be located in the segment's `data\INI` subdirectory.

Except for COTS segments, segments shall not create root keys, but may create subkeys underneath the root keys as desired. In all cases, segments shall create segment subkeys underneath

```
HKEY_LOCAL_MACHINE\SOFTWARE\COE
```

using the convention ***SegType**SegDirName*** where ***SegType*** is one of the following:

`Account Groups`	for account group segments
`COE`	for COE-component segments
`COTS`	for COTS products
`Data`	for data segments
`Patches`	for patch segments
`Software`	for all other segment types.

SegDirName is the segment's directory name. Segments shall use the segment prefix to name all registry subkey entries.

For example, assume a software segment whose directory is `SegA` has a segment prefix `SEGA`. Assume the segment needs to store two pieces of information underneath `HKEY_LOCAL_MACHINE\SOFTWARE`:

1. the last coordinate system used (UTM, Lat/Lon, etc.) and
2. the last time a certain parameter was computed.

Then the required registry path is

[6] Developers should avoid overuse of the NT registry. It is best used for system-level constructs and *not* as a total replacement for `.INI` files.

```
HKEY_LOCAL_MACHINE\SOFTWARE\COE\Software\SegA
```

and two appropriately named subkeys underneath this entry for storing value entries are `SEGA_Last_Coord` and `SEGA_Last_Time`.

> **Note:** The key `HKEY_LOCAL_MACHINE\SOFTWARE\COE` is created when the DII COE is installed.

Microsoft encourages use of the registry in some ways that are strictly forbidden in the COE because the `COEInstaller` tool performs some of these actions automatically. Refer to the subsection below on segment installation for actions performed by the installer. Segments, excepting COTS segments, shall not use the registry to duplicate any actions performed by the COE installation software:

- Segments shall *not* register "uninstall" information in the `Uninstall` key beneath `CurrentVersion`, with two exceptions: (1) when the segment is a COTS product that does register "uninstall" information as part of its setup, or (2) as authorized by the DII COE Chief Engineer. If the segment does register "uninstall" information, it shall specify the `$USES_UNINSTALL` keyword in the `Direct` descriptor.

- Segments shall use the `Processes` descriptor to specify background processes. Segments shall *not* add values to either the `Run` or `RunOnce` keys beneath the `CurrentVersion` key. The segment shall use the `$RUN_ONCE` keyword to specify the requirement to run certain executables the next time, and only the next time, the system is restarted. Use of this keyword requires approval by the cognizant Chief Engineer.

6.5 Reserved Prefixes, Symbols, and Files

The segment prefixes listed as reserved in Chapter 5 are also reserved in the NT-based COE. The following segment prefixes are reserved and are specific to the NT-based COE:

NT	Generic NT segments
WIN	Generic Windows segments
WIN95	Windows 95 segments
WINNT	Windows NT segment for 80x86 platforms

The environment variables listed as reserved in Chapter 5 are also reserved in the NT-based COE. Segments shall not create environment variables with the same name as any reserved environment variable. The following have no meaning in the NT-based COE, and are not guaranteed to be set:

```
DISPLAY
LD_LIBRARY_PATH
SHELL
TERM
TZ
XAPPLRESDIR
XENVIRONMENT
XFONTSDIR
```

All remaining environment variables listed in Chapter 5 are also defined for the NT-based COE.

The root-level AUTOEXEC.BAT and CONFIG.SYS files are reserved files and shall not be modified by any segment, excepting COTS segments. Moreover, all windows INI files (specifically, WIN.INI and SYSTEM.INI) are reserved files and shall not be modified by any segment, excepting COTS segments. Segments should create and modify their own local INI files.

6.6 Programming Standards

Programming in the Windows environment is considerably different from the Unix/X Windows environment. This subsection details programming practices that are required to minimize problems in mixing the two environments.

6.6.1 File System

Windows NT supports five file systems: FAT, VFAT, HPFS, NTFS, and CDFS. FAT (File Allocation Table) is the file system used by MS-DOS, but it is extended in both Windows 95 and Windows NT (version 3.5 and later) to support long filenames (e.g., VFAT). HPFS (High Performance File System) originated with OS/2®. NTFS (NT File System) originated with Windows NT as an improvement over both HPFS and FAT. CDFS (CD-ROM File System) is specific to CD-ROM devices.

NTFS is the file system required for the DII COE because its security architecture corrects known problems in FAT. DII-compliant systems shall be formatted to use NTFS. However, the FAT and VFAT file systems are the only available file systems for floppy disks. Therefore, the COE requires NTFS for hard disk drives, but supports FAT and VFAT for floppy drives. The type of file system in use should be transparent to most segments. When there is a choice, NTFS shall be used for hard and VFAT shall be used for floppy drives.

A further complication is that NTFS filenames use the 16-bit *Unicode*® character set instead of 8-bit ASCII. Unicode is a technique for representing foreign alphabets (Japanese kanji, Chinese bopomofo, Greek, etc.). NT segments are not required to create Unicode strings, but segments must be able to read filenames that may be Unicode strings. This requirement is necessary because commercial products may be distributed on media that uses Unicode filenames, and because Windows NT uses Unicode strings internally. Windows 95 does not; it uses 8-bit ASCII strings internally.

Pathnames in Windows usually include a disk drive designation (e.g., `C:`). The disk drive containing the desired file may be located remotely on another machine. Windows allows symbolic names, called the *Universal Naming Convention* (UNC), to be given to remote paths so that an application need not know the workstation, disk drive, nor exact path to reach a particular file. UNC pathnames start with two backslashes (`\\`) followed by the server name, followed by the desired pathname and filename. Segments shall support the use of UNC pathnames.

To summarize,

1. Segments shall support the use of long filenames. Filenames are not allowed to contain embedded spaces, and should use file extensions as appropriate to conform to standard Windows usage.

2. Segments shall support use of UNC filenames.

3. Segments shall be Unicode aware, particularly with regard to filenames.

6.6.2 Dynamic Link Libraries

NT segments shall use *dynamic link libraries* (DLLs) to the maximum extent feasible. DLLs are located in the segment's `bin` subdirectory, except for COE segments. COE DLLs are located underneath the directory `\h\COE\bin` for all COE segments.

Windows originally exported DLL functions by assigning ordinal numbers to each exported function. Modules linked to DLL functions by ordinal number. However, later versions allowed linking to be by symbolic name rather than ordinal numbers. All NT segments shall link by symbolic name, and shall export DLL functions by symbolic name rather than ordinal numbers. The reason for this requirement is that ordinal numbers could change over time for exported functions, whereas the symbolic name will not.

6.6.3 Graphics

PC segments shall support VGA and SVGA resolutions, and should use the Win32 API Graphics Display Interface (GDI) for creation of 2D graphics. This interface handles all calls made by applications for graphic operations and thus provides a standard interface for such calls. As a result, the Win32 GDI allows segments to be developed which are independent of the type of graphics output device in the end user's system. That is, segments need only make calls to standard graphic services provided by the Win32 subsystem regardless of the display, printer, or multi-media hardware used in the system.

To improve 2D graphics performance, the WinG library may be used. WinG is an optimized library designed to enable high-performance graphics techniques under Win32, Windows NT, Windows 95, and future Windows releases. Segments should use OpenGL APIs for 3D graphics. OpenGL is a software interface that allows the creation of high-quality 3D color images complete with shading, lighting, and other effects. OpenGL is an open standard designed to run on a variety of computers and a variety of operating systems. It consists of a library of API functions for performing 3D drawing and rendering.

6.6.4 Fonts

Windows supports three different kinds of font technologies to display and print text: raster, vector, and TrueType®. The differences between these fonts reflect the way that the *glyph* for each character or symbol is stored in the respective font resource file. In raster fonts, a glyph is a bitmap that Windows uses to draw a single character or symbol in the font. In vector fonts, a glyph is a collection of line endpoints that define the line segments Windows uses to draw a character or symbol in the font. In TrueType fonts, a glyph is a collection of line and curve commands as well as a collection of hints. Windows uses the line and curve commands to define the outline of the bitmap for a character or symbol in the TrueType font. Windows uses the hints to adjust the length of the lines and shapes of the curves used to draw the character or symbol. These hints and the respective

adjustments are based on the amount of scaling used to reduce or increase the size of the bitmap.

Vector and TrueType fonts are device independent, while raster fonts are not. TrueType fonts provide both relatively fast drawing speed and true device independence. By using the hints associated with a glyph, application software can scale the characters from a TrueType font up or down and still maintain their original shape. Segments shall use TrueType fonts to take advantage of the increased performance, flexibility, and WYSIWYG (What-You-See-Is-What-You-Get) screen to printer characteristics. Custom application-specific fonts shall be avoided in favor of using industry standard fonts wherever possible.

6.6.5 Printing

NT segments shall use the built in printing facilities provided by Windows. This includes using the Windows supplied printer common dialog box for configuring a printer, selecting print quality, selecting the number of copies, etc. All access to the printer shall be through Windows APIs.

Developers should be aware that some Win32 APIs are available only in Windows NT. Developers may use these APIs, but should ensure that the segment still operates correctly in a Windows 95 environment. As appropriate, NT segments should support drag and drop printing.

6.6.6 Network Considerations

UNC Filenames

NT segments shall support UNC filenames to access network shared drives and directories. If necessary, a segment can use the WinNet APIs to determine if a pathname is a network pathname.

The COE contains three pre-defined shared directories: `\h\data\PCglobal`, `\h\data\global`, and `\h\USERS\global`. The proper UNC filename to use for these three directories is determined by accessing registry subkeys underneath `HKEY_LOCAL_MACHINE\HARDWARE` as follows:

```
COE\Shared\data_PCglobal    \h\data\PCglobal
COE\Shared\data_global      \h\data\global
COE\Shared\USERS_global     \h\USERS\global
```

NT segments that create network sharable services or devices shall store UNC information in the registry. The subkey shall be either `COE\Shared` or `SEGS\Shared` depending upon segment type. The subkey shall be located underneath `HKEY_LOCAL_MACHINE\HARDWARE` for hardware devices (e.g., disk drives) or `HKEY_LOCAL_MACHINE\SOFTWARE` for software (e.g., servers). The segment shall document the proper registry information in the API documentation for the segment.

Network Byte Ordering

Computer architectures sometimes differ in the convention they use for how bytes are ordered in a word. This is the so-called "*big-endian, little-endian*" problem. Computers in which the *most* significant byte in a word is the leftmost byte use big-endian byte ordering. Computers in which the *least* significant byte in a word is the leftmost byte use little-endian byte ordering. Intel architectures use little endian byte ordering. When data is sent across the network, it is important to agree upon the same convention for byte ordering. The big-endian convention is also known as the *network byte order* and has been established as the industry standard.

The COE adopts the industry standard for byte ordering[7] and requires the use of network byte order for any data transmitted across a heterogeneous LAN. Segments shall ensure that all network data is transmitted in network byte order, except for certain data accessed on a PC-only network shared disk drive such as the `PCglobal` data directory. Segments shall use APIs in the WinSock interface to ensure that data sent across the network is in network byte order. Segments shall store disk data accessible only by PCs in native PC byte order, but shall store disk data accessible by non-PCs in network byte order. The shared data directories and byte ordering are as follows:

`\h\data\PCglobal`	PC native byte order. Data here is shared, but is restricted to only PCs.
`\h\data\global`	Network byte order. Data in this directory may be accessible from a Unix platform as well as PCs.
`\h\USERS\PC`	Native byte order. Data located here is specific to operator login accounts. Since a login account is either for Unix or a PC but never both, this data is platform-specific.

Network Communications

Windows NT supports four transport layer protocols:

NetBEUI	provides compatibility with existing LAN Manager, LAN Server, and MS-Net installations.
TCP/IP	provides compatibility with standard Unix environments and a routable protocol for wide area networks.

[7] DCE developers should use DCE functions to implement network byte ordering. All other developers should use XDR protocol.

Data Link Control (DLC)	provides an interface for access to mainframes and printers attached to networks.
AppleTalk®	provides interoperability with Macintosh networks.

TCP/IP is the COE standard network protocol. Segments shall perform network communications through WinSock APIs. Communications shall be designed to operate asynchronously to ensure that the server or application does not "hang" while waiting for a response.

6.6.7 Miscellaneous

The following statements apply to all new segment development. COTS segments may not meet all mandatory requirements, but shall be documented where they do not fulfill a mandatory requirement. To the extent possible, segments should conform to the requirements stipulated by Microsoft for allowing an application to use the Windows Logo.

Mandatory

1. All hardware shall be NT-compliant, as defined by the document *Microsoft Windows NT Hardware Compatibility List #4094*.

2. Segments shall support VGA and SVGA graphics.

3. Segments shall be "close aware." This means that the segment must enable the Close command and periodically check the close flag through the Query Close function.

4. Segments shall use common control and common dialog functions contained in `COMCTL32.DLL` and `COMDLG32.DLL`.

5. As appropriate, segments shall support cut and paste operations through the clipboard.

6. As appropriate, segments shall support drag and drop operations.

7. Segments shall support 16x16, 32x32, and 64x64 icons.

8. Segments shall *not* use MS-DOS APIs inside a compiled program. These functions are typically interrupt-driven or depended upon specific memory addresses and are not portable. Win32 APIs only are to be used within a compiled program. Segments may use MS-DOS commands within the various installation-related batch files.

9. Segments shall use *only* Win32 APIs. Win16 APIs are not supported and shall not be used unless they are part of a COTS product for which there is no 32-bit alternative.

10. Segments shall not duplicate functionality already provided by Windows.

11. Segments shall support long filenames and UNC.

12. Segments shall be Unicode aware.

Optional

1. Segments should run the Windows SDK tool `PORTTOOL.EXE` to identify potential problems with how Windows APIs are being used.

2. Segments should operate under both Windows NT and Windows 95. The segment should degrade gracefully if it uses APIs found only in Windows 95 while running in a Windows NT environment, and vice versa.

3. Segments should define the `STRICT` constant when compiling Windows code. This enables strict type checking during compilation.

4. Segments should avoid using environment variables. The registry or local INI files are preferred alternatives.

5. Developers are encouraged to use message crackers contained in `WINDOWSX.H`. Message crackers are a set of macros that makes code more readable, simplifies porting, and reduces the need to do type casting.

6. As appropriate, segments should register icons for document types and provide a viewer to allow the shell to display them. This is done through the `HKEY_CLASSES_ROOT` registry. Refer to Microsoft documentation for the required procedures. A future COE release may provide segment descriptors to accomplish this.

6.7 Segment Installation

Segment installation follows the same sequence as for the Unix environment, and is defined in Chapter 5. The key

```
HKEY_LOCAL_MACHINE\SOFTWARE\COE
```

is automatically created when the DII COE kernel is loaded. As segments are installed on the NT platform, `COEInstaller` creates registry entries under this key corresponding to segment type as explained in subsection 6.4. That is, assuming *`SegDir`* is the segment's directory name and *`SegType`* is the segment's type, the installer creates the following registry key entry:

```
HKEY_LOCAL_MACHINE\SOFTWARE\COE\SegType\SegDir
```

All entries underneath this registry key are deleted automatically when the segment is deleted.

`COEInstaller` sets the environment variables `INSTALL_DIR`, `MACHINE_CPU`, and `MACHINE_OS` for use in the `PreInstall.BAT` (or `.EXE`) and `PostInstall.BAT` (or `.EXE`) descriptors. `SYSTEM_ROOT` is set to indicate where Windows was installed. The installer also stores the location where the segment was loaded in the subkey *`SegDir`*`\SegPath`. The value of this subkey includes the disk drive where the segment was loaded, but it cannot be accessed until after segment loading is completed.

Segments may use commercially available programs to create "`setup.exe`" files, but this is **strongly** discouraged. The `setup.exe` file must be invoked from within the segment's `PostInstall` program. Moreover, the installation program must obey the constraints outlined in the *I&RTS*. In particular, segments shall not perform any operations (e.g., create registry entries, uninstall a segment) that the COE installation software performs because these will be performed whether or not a commercial installation package is used. It is strongly recommended that segments use the segment descriptors provided to "self-describe" the segment and allow the COEInstaller to perform the installation chores. This ensures a consistent approach for all segment installations, and avoids potential conflicts between different segment installation approaches.

6.8 NT COE Descriptors

The descriptor files defined in Chapter 5 apply to the NT-based COE as well. This section is provided as a quick reference for items that are NT-related. Refer to Chapter 5 for complete discussion of each of the descriptors discussed below.

General comments follow.

- NT segments are required to use `SegInfo` for descriptors; that is, NT segments may not use individual descriptor files since these are obsolete. All obsolete conventions are explicitly invalid for NT segments and are flagged as errors by `VerifySeg`.

- When a home directory must be given, such as when specifying segment dependencies, a default disk drive may also be specified. If no default is specified, disk drive `C` is assumed. Pathnames must be given using '\' in conformance to the Windows environment.

- When a disk drive designation is given, it and any associated pathname must be enclosed in double quotes. This is required so that the tools can distinguish between use of ':' as a field delimiter for descriptor lines, or as a separator between a disk drive name and a directory pathname. Disk drive designations are assumed to be advisory only. The COE will attempt to satisfy the request for the disk drive given. If this fails, the COE will attempt to satisfy the request on another disk drive until all hard disk drives have been attempted, or the request is satisfied.

  For example, a `Requires` descriptor entry for a segment on drive `D` under the directory `\h\SegA` would be described as

  ```
  segname:prefix:"D:\h\SegA":[version{:patch}]
  ```

  The COE will attempt to find the required segment on drive `D:` in directory `\h\SegA`. If the required segment is not there, the COE will search[8] for the segment on other disk drives.

- In accordance with commercial standards, executable descriptors shall have either a `.EXE` extension (for compiled programs) or a `.BAT` extension (for batch files). This applies to the "scripts" used in the installation process: `DEINSTALL`, `PostInstall`, `PreInstall`, and `PreMakeInst`. Segment descriptor files may optionally have a `.TXT` extension.

[8] The search description here should not be taken too literally. The COE may in fact use the registry and not actually perform a physical search of any disk drive. The point being made here is that designation of a disk drive is allowed, but it is considered to be advisory only.

- The ${SYSTEM_ROOT} environment variable is set to indicate where the Windows system directory is located. This environment variable may be used in the installation-related "scripts" at install time.

Comments related to specific descriptors follow.

AcctGroup

NT account groups must omit the *shell* parameter. It has no meaning in Windows.

COEServices

The $GROUPS and $PASSWORDS keywords are not supported for NT platforms. VerifySeg generates a warning if a segment descriptor contains these keywords.

DEINSTALL.EXE and DEINSTALL.BAT

Chapter 5 indicates that DEINSTALL is executed prior to a segment being removed from the system. A segment that does not include a DEINSTALL descriptor is a permanent segment and may be updated, but not removed. In many situations, it is desirable for the segment to be removable, but there are no actions that DEINSTALL must perform. For this reason, the NT-based COE allows DEINSTALL to exist as a zero-length file.

FileAttribs

Because file permissions are different between the Unix and NT environments, FileAttribs is operating system specific. The COE tool MakeAttribs, when run on an NT platform, will create a proper FileAttribs file for NT segments. C style #ifdef preprocessor statements may be used to combine a Unix and NT FileAttribs descriptor.

Hardware

The *diskname* field for the $PARTITION keyword must be a disk drive name. For example, to indicate that a segment requires 20MB on the F disk drive, the proper $PARTITION statement is

```
$PARTITION:"F:":20480
```

Network

The Network descriptor is not presently supported for NT platforms. VerifySeg will issue a warning if a Network descriptor is found for an NT segment.

Processes

The `$RUN_ONCE` keyword identifies process that should be run the next time the system is started. This keyword requires authorization by the cognizant Chief Engineer because of potential security and performance risks.

Registry

The `Registry` descriptor allows the segment to have the COEInstaller create registry key entries.

ReqrdScripts

Environment extension files are not supported for NT platforms. Therefore, the `ReqrdScripts` descriptor is not supported for NT platforms. `VerifySeg` will print a warning if this descriptor is present.

Requires

The pathname given for the `$HOME_DIR` keyword may include a disk drive designation. Because a disk drive designation is considered advisory, it should generally not be given. If a disk drive is not specified, the installation tools will load the segment in the directory indicated, but on the first available disk drive.

SegName

The `$COMPANY_NAME` and `$PRODUCT_NAME` keywords allow a COTS segment to specify company and product names for the registry. These are added by the `COEInstaller`, and must not be specified if the COTS product creates registry entries itself.

SharedFile

This descriptor allows the segment to identify shared DLLs.